第6章 树和二叉树

杭州電子科技大学
HANGZHOU DIANZI UNIVERSITY

6.1 树和二叉树的定义

- 案例引入

6.2 二叉树的抽象数据类型定义

6.3 二叉树的性质和存储结构

6.4 遍历二叉树和线索二叉树

6.5 树和森林

6.6 哈夫曼树及其应用

6.6 案例分析与实现

《数据结构》

后序遍历二叉树的操作定义

杭州電子科技大学 HANGZHOU DIANZI UNIVERSITY

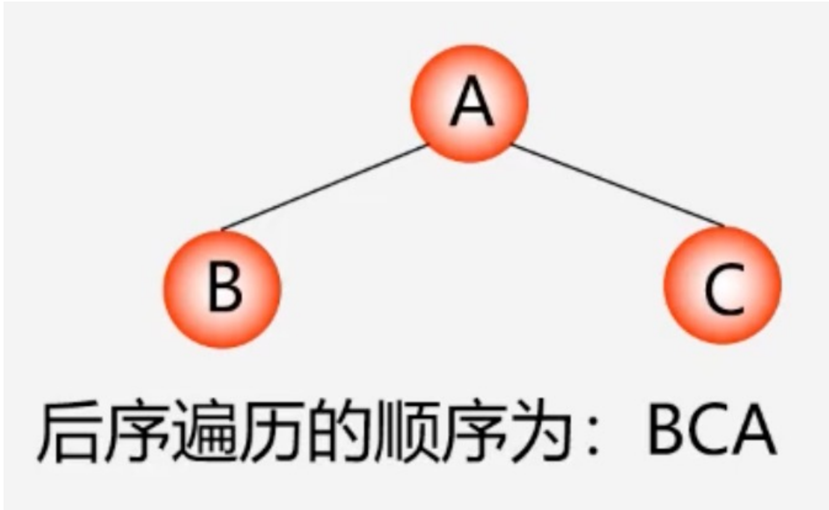
- 后序遍历过程：

若二叉树为空，则空操作；否则

① 后序遍历其左子树；

② 后序遍历其右子树。

③ 访问根结点；



后序遍历二叉树的操作定义

杭州電子科技大学 HANGZHOU DIANZI UNIVERSITY

(D E F B) (H G I C) A

后序遍历=> D E F B H G I C A

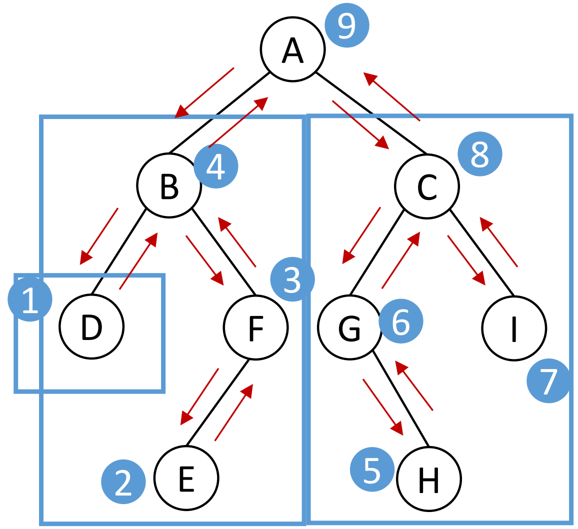
➢ 后序遍历过程：

若二叉树为空，则空操作；否则

① 后序遍历其左子树；

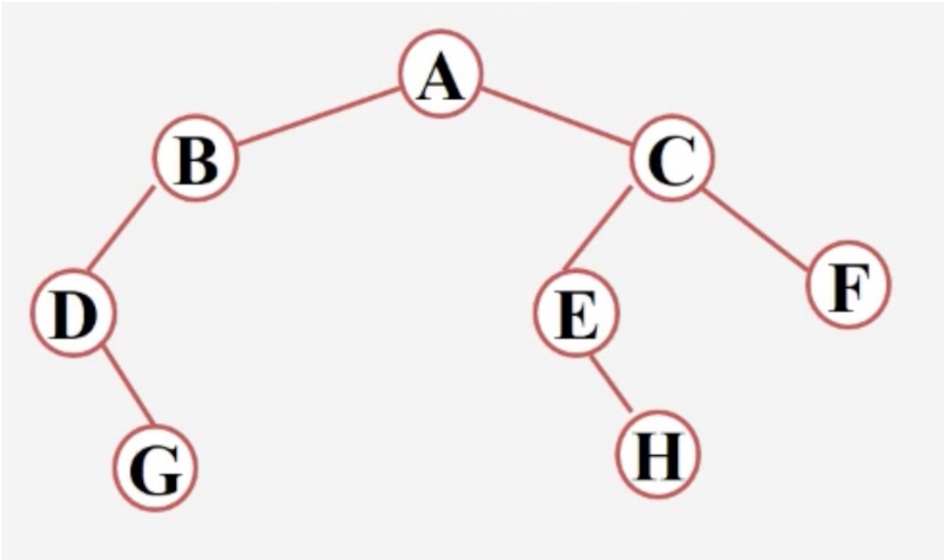
② 后序遍历其右子树。

③ 访问根结点；



例题

写出下图二叉树的各种遍历顺序



先序： A B D G C E H F

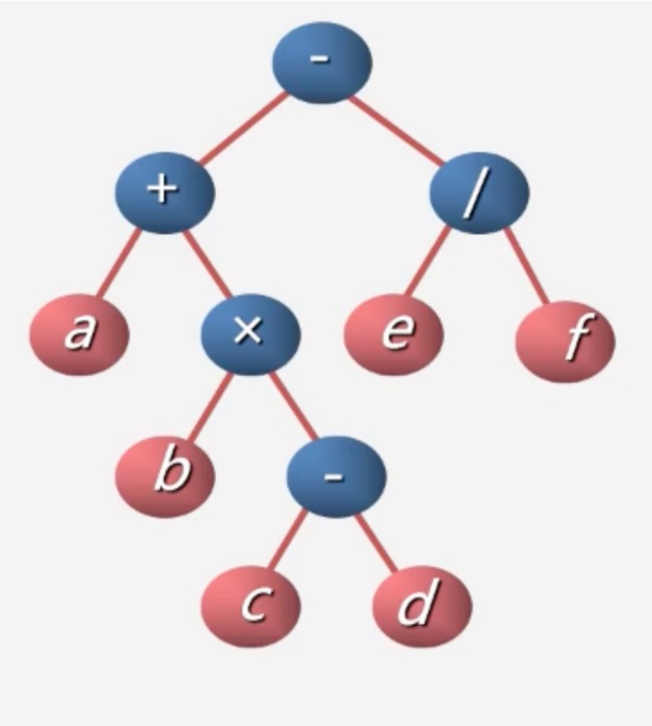
中序： D G B A E H C F

后序： G D B H E F C A

例题——用二叉树表示算术表达式

杭州電子科技大学 HANGZHOU DIANZI UNIVERSITY

请写出下图所示二叉树的先序、中序和后序遍历顺序



遍历结果：

先序：-+axb-cd/ef（表达式的前缀表示（波兰式））

中序：a+bxc-d-e/f （表达式的中缀表示）

后序：abcd-x+ef/- （表达式的后缀表示 (逆波兰式) ）

2. 根据遍历序列确定二叉树

- 若二叉树中各结点的值均不相同，则二叉树结点的先序序列、中序序列和后序序列都是唯一的

- 由二叉树的先序序列和中序序列，或由二叉树的后序序列和中序序列可以确定唯一一棵二叉树

例题——已知先序和中序序列求二叉树

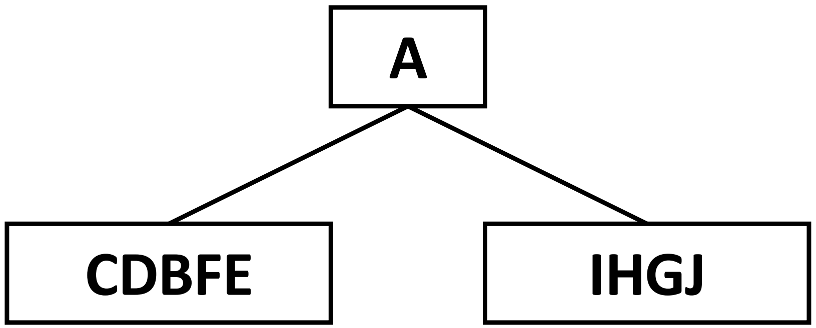
例：已知二叉树的先序和中序序列，构造出相应的二叉树

先序： A B C D E F G H I J

中序： C D B F E A I H G J

分析：由先序序列确定根；由中序序列确定左右子树

解：1、由先序知根为A，则由中序知左子树为CDBFE，右子树为IHGJ



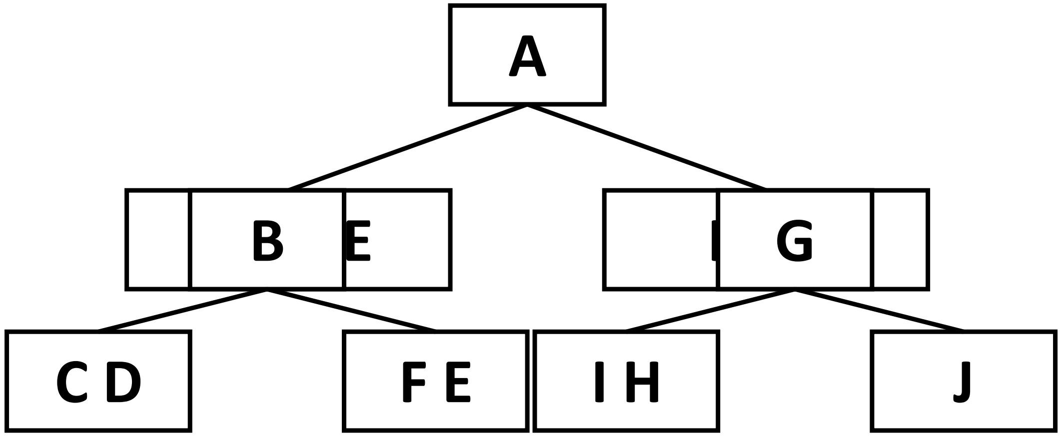
例题——已知先序和中序序列求二叉树

杭州电子科技大学 HANGZHOU DIANZI UNIVERSITY

2、再分别在左、右子树的序列中找出根、左子树序列、右子树序列。

先序：A B C D E F G H I J

中序：C D B F E A I H G J

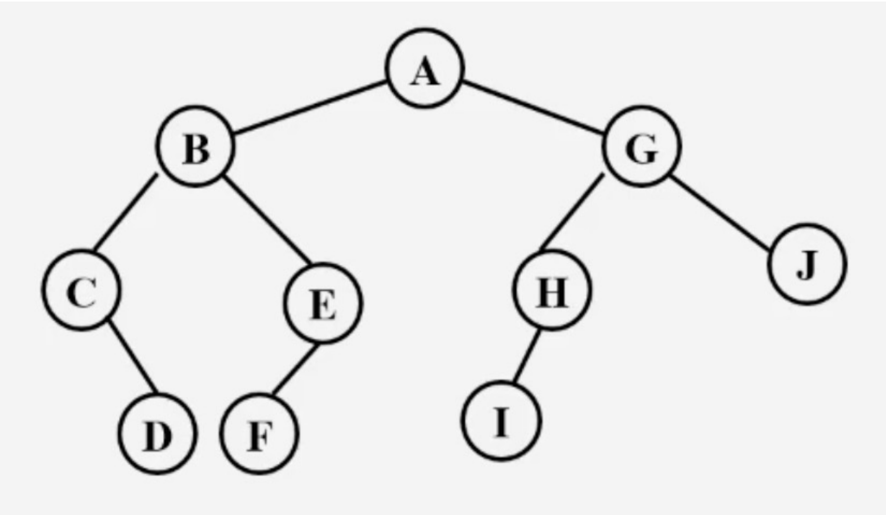


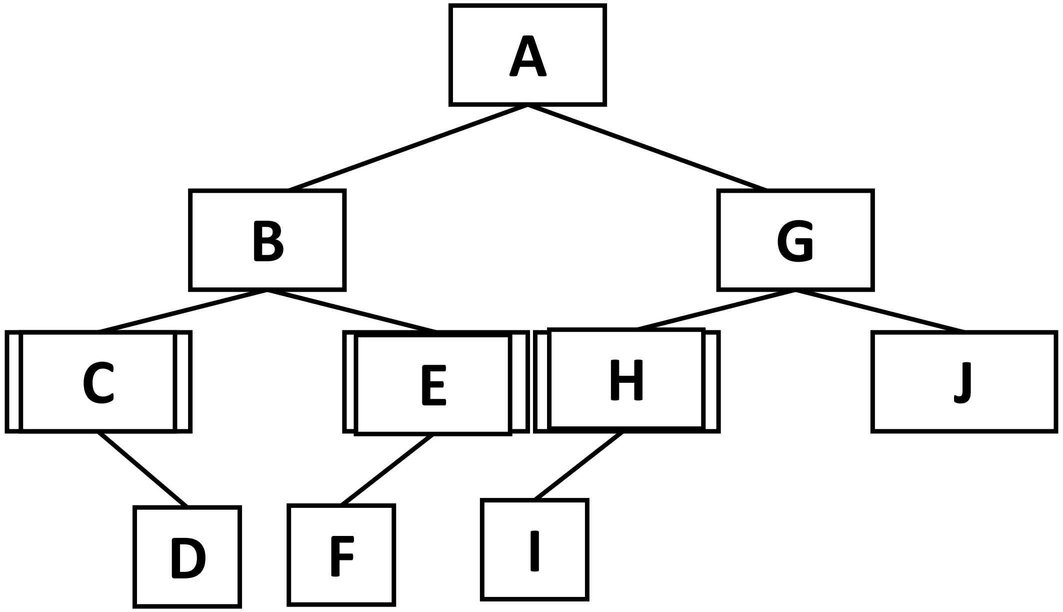
例题——已知先序和中序序列求二叉树

3、以此类推，直到得到二叉树

先序：A B C D E F G H I J

中序：C D B F E A I H G J





例题——已知中序和后序序列求二叉树

杭州電子科技大學 HANGZHOU DIANZI UNIVERSITY

实例分析：已知一颗二叉树的

中序序列：BDCEAFHG

后序序列：DECBHGFA，请画出这颗二叉树

提示：

后序遍历，根结点必在后序序列尾部

后序：DECBHGFA

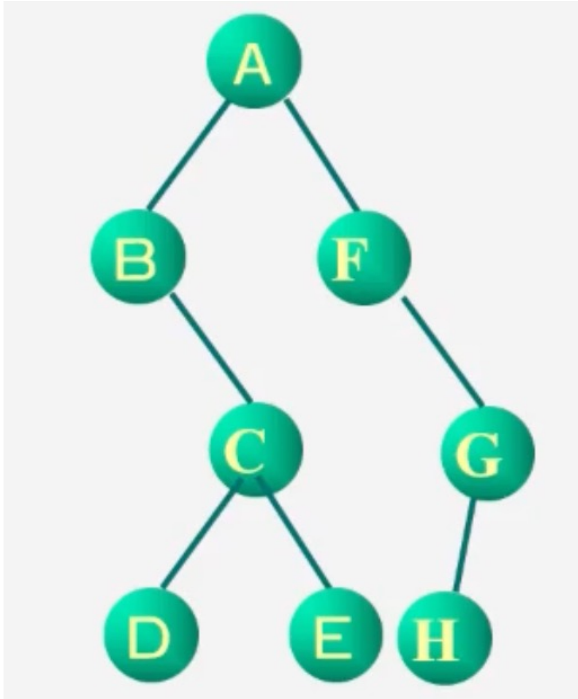
中序：BDCEAFHG

后序：DECB　　后序：DEC

中序：BDCE　　中序：DCE

后序：HGF　　后序：HG

中序：FHG　　中序：HG



6.4 遍历的算法实现 – 先序遍历

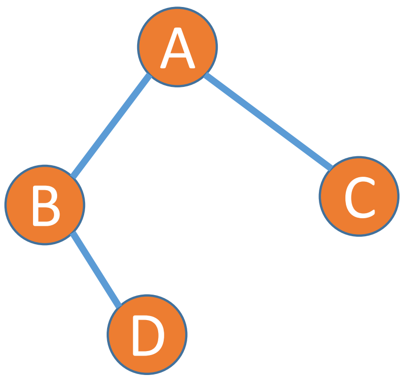
若二叉树为空，则空操作；

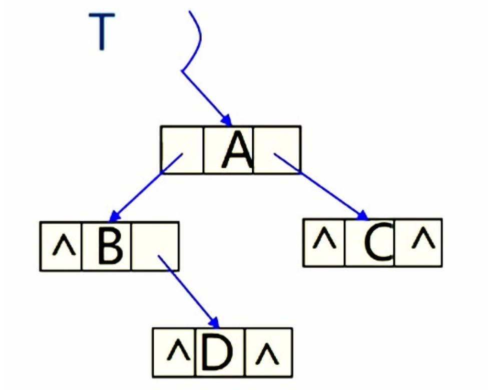
若二叉树非空，

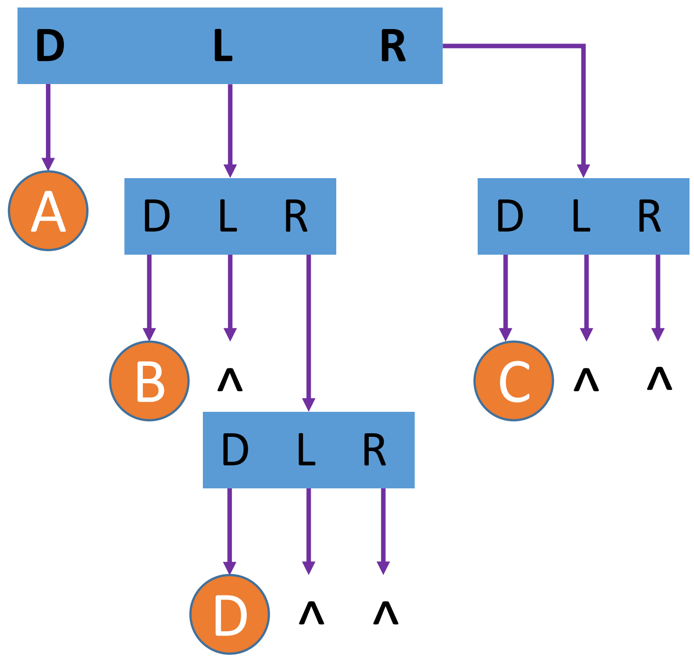
访问根结点（D）

前序遍历左子树（L）

前序遍历右子树（R）







先序遍历序列：A B D C

6.4 二叉树先序遍历算法

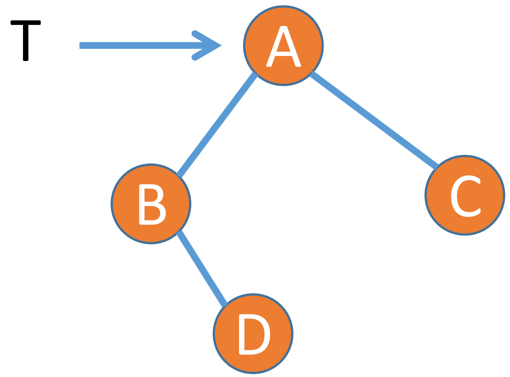
杭州電子科技大學 HANGZHOU DIANZI UNIVERSITY

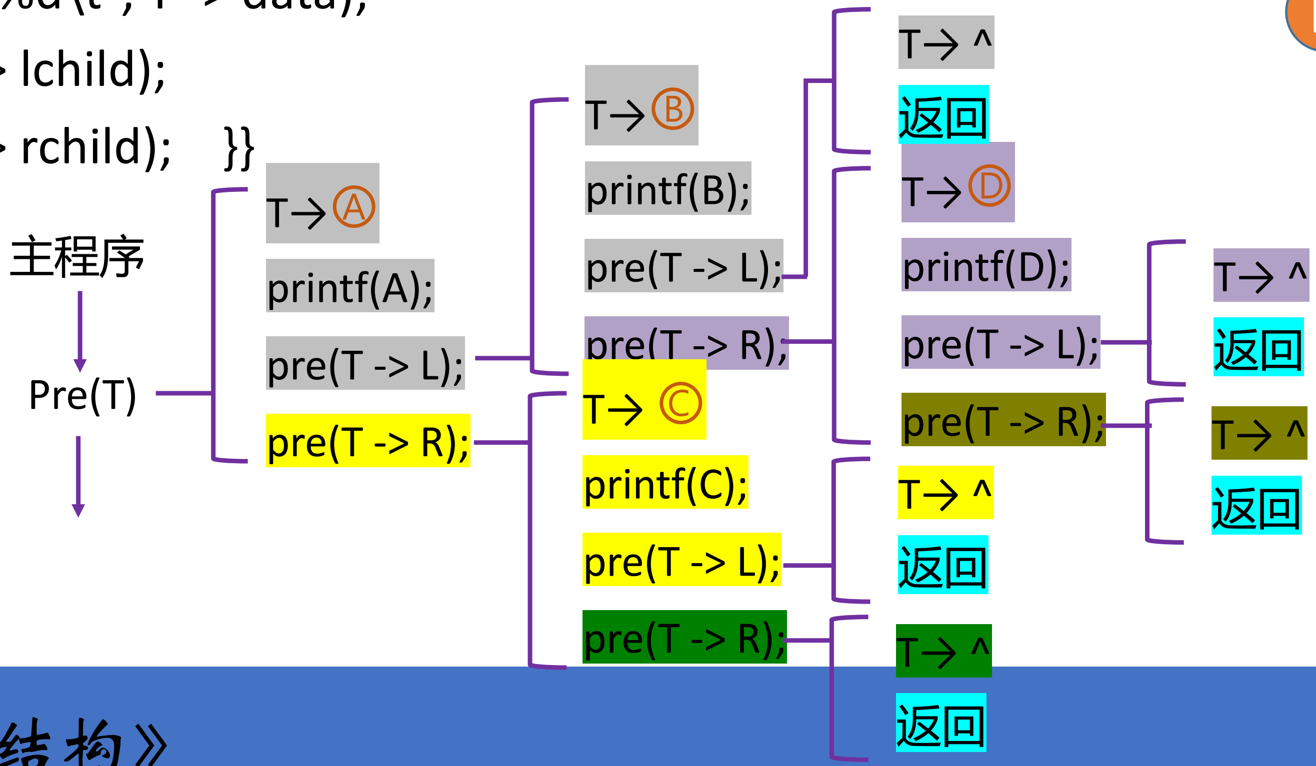
```
Status PreOrderTraverse(BiTree T){
    if(T == NULL) return OK;    //空二叉树
    else{
        visit(T);    //访问根节点
        PreOrderTraverse(T -> lchild);    //递归遍历左子树
        PreOrderTraverse(T -> rchild);    //递归遍历右子树
    }
}
```

例如，输出根节点 printf(“%d\t”, T->data);

6.4 二叉树先序遍历算法


```
void Pre(BiTree *T){
  if(T != NULL){
    printf("%d\t", T -> data);
    pre(T -> lchild);
    pre(T -> rchild);    }}
```





6.4 遍历的算法实现 – 中序遍历

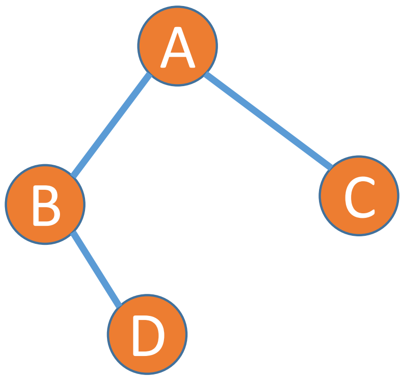
若二叉树为空，则空操作；

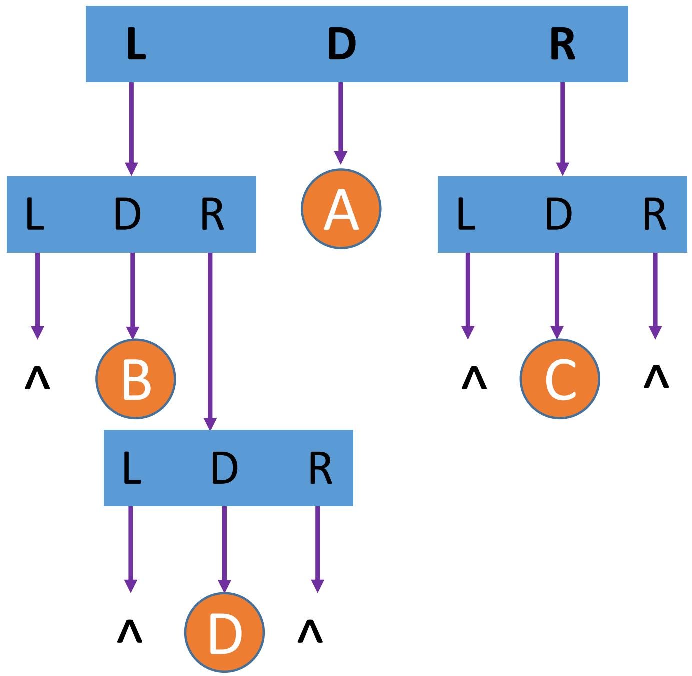
若二叉树非空，

中序遍历左子树（L）

访问根节点（D）

中序遍历右子树（R）





中序遍历序列：B D A C

6.4 二叉树中序遍历算法


```
Status InOrderTraverse(BiTree T){
    if(T == NULL) return OK;    //空二叉树
    else{
        InOrderTraverse(T -> lchild);    //递归遍历左子树
        visit(T);    //访问根节点
        InOrderTraverse(T -> rchild);    //递归遍历右子树
    }
}
```

6.4 遍历的算法实现 – 后序遍历

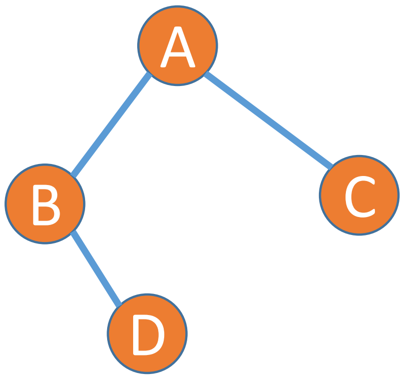
若二叉树为空，则空操作；

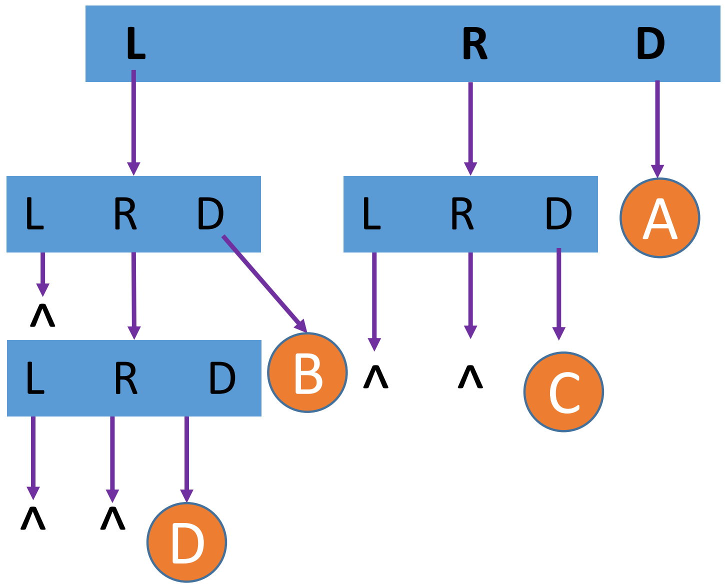
若二叉树非空，

后序遍历左子树（L）

后序遍历右子树（R）

访问根节点（D）





后序遍历序列：D B C A

6.4 二叉树后序遍历算法

杭州电子科技大学 HANGZHOU DIANZI UNIVERSITY

```
Status PostOrderTraverse(BiTree T){
    if(T == NULL) return OK;    //空二叉树
    else{
        PostOrderTraverse(T -> lchild);    //递归遍历左子树
        PostOrderTraverse(T -> rchild);    //递归遍历右子树
        visit(T);    //访问根节点
}
}
```

6.4 遍历算法的分析


```
Status PreOrderTraverse(BiTree T){  //前序
   if(T == NULL) return OK;
   else{
      visit(T);
      PreOrderTraverse(T -> lchild);
      PreOrderTraverse(T -> rchild); }
}
```

```
Status PostOrderTraverse(BiTree T){   //后序
   if(T == NULL) return OK;
   else{
      PostOrderTraverse(T -> lchild);
      PostOrderTraverse(T -> rchild);
       visit(T);
    }
}
```

```
Status InOrderTraverse(BiTree T){  //中序
   if(T == NULL) return OK;
   else{
      InOrderTraverse(T -> lchild);
      visit(T);
      InOrderTraverse(T -> rchild);
   }
}
```

6.4 遍历算法的分析

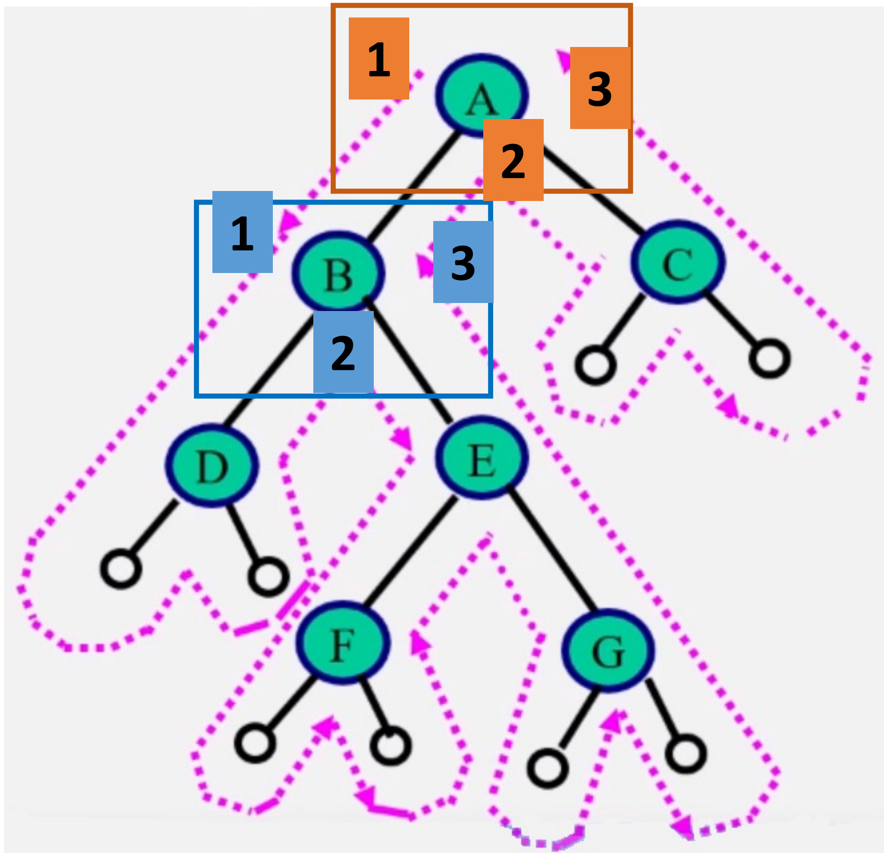
如果去掉输出语句，从递归的角度看，三种算法是完全相同的，或说这三种算法的访问路径是相同的，只是访问结点的时机不同。

从虚线的出发点到终点的路径上，每个结点经过3次

第1次经过时访问 = 先序遍历

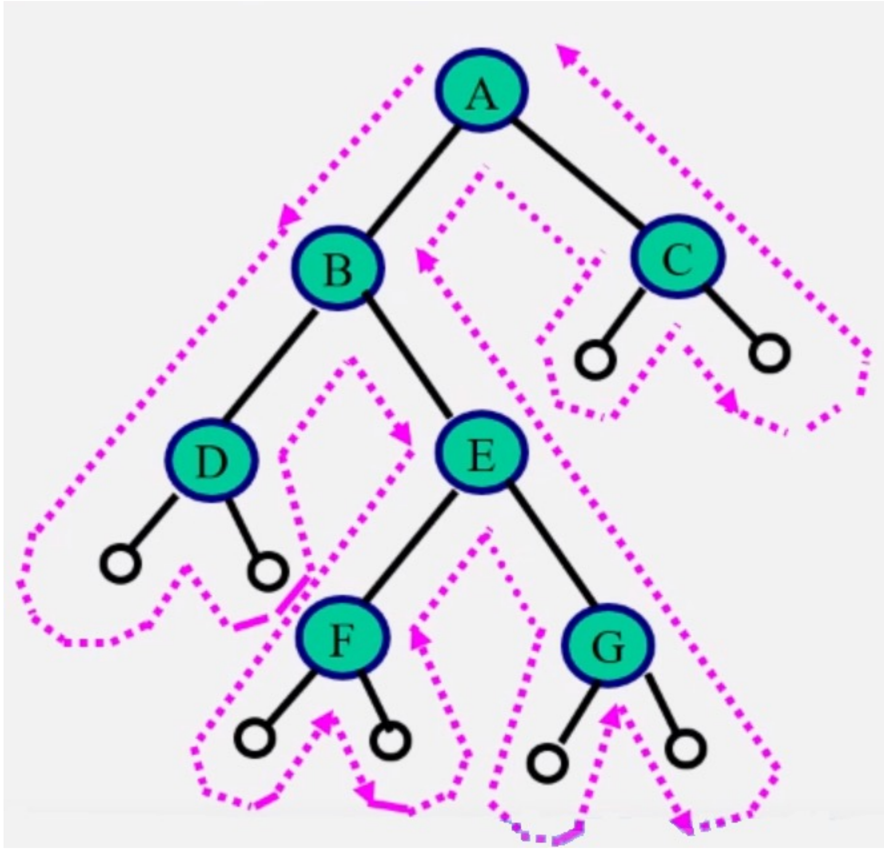
第2次经过时访问 = 中序遍历

第3次经过时访问 = 后序遍历



6.4 遍历算法的分析

- 时间效率： O(n) //每个结点只访问一次
- 空间效率： O(n) //栈占用的最大辅助空间



6.4 遍历二叉树的非递归算法

杭州電子科技大学 HANGZHOU DIANZI UNIVERSITY

中序遍历非递归算法

二叉树中序遍历的非递归算法的关键：在中序遍历过某结点的整个左子树后，如何找到该结点的根以及右子树。

基本思想：

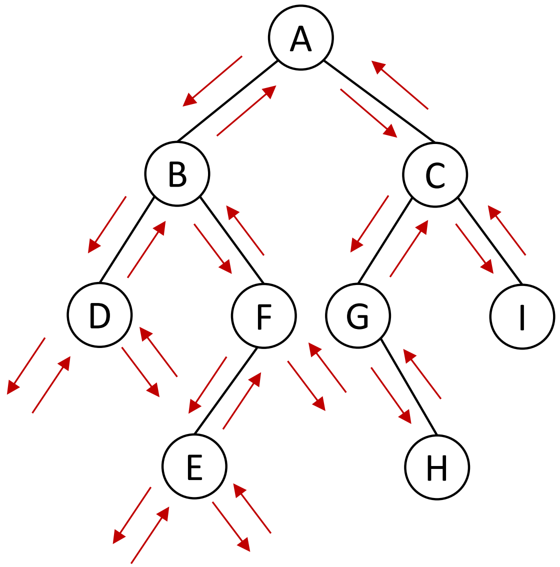
（1）建立一个栈

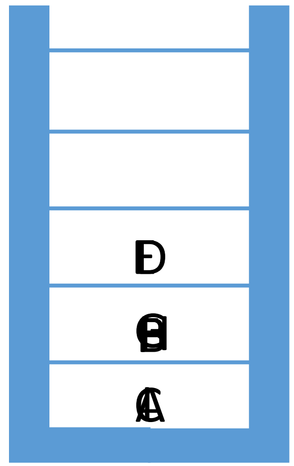
（2）根结点进栈，遍历左子树

（3）根结点出栈，输出根结点，遍历右子树

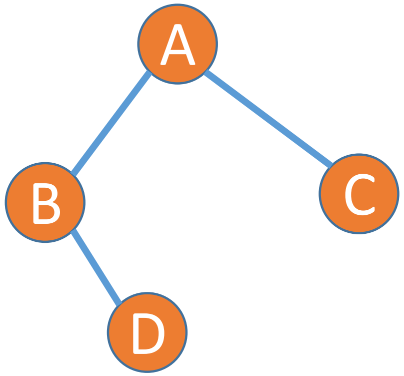
6.4 中序遍历的非递归操作演示

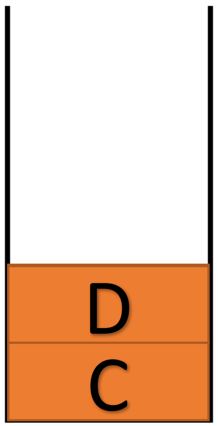
杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY





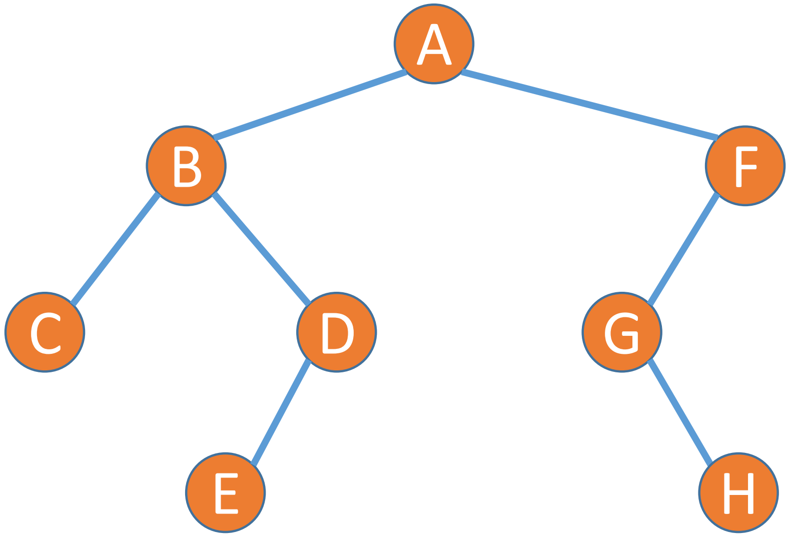
6.4 中序遍历的非递归算法






```
Status InOrderTraverse(BiTree T){
    BiTree p;    InitStack(S);    p = T;
    while(p || !StackEmpty(S))    {
        if(p)    {Push(S, p);    p = p -> lchild;}
        else    {Pop(S, q);    printf("%c", q -> data);
                        p = q -> rchild;    }
        }//while
    return OK;
}
```

6.4 二叉树的层次遍历



层次遍历结果： ABFCDGEH

对于一颗二叉树，从根结点开始，按从上到下、从左到右的顺序访问每一个结点。每一个结点仅仅访问一次

6.4 二叉树的层次遍历

算法设计思路：　使用一个队列

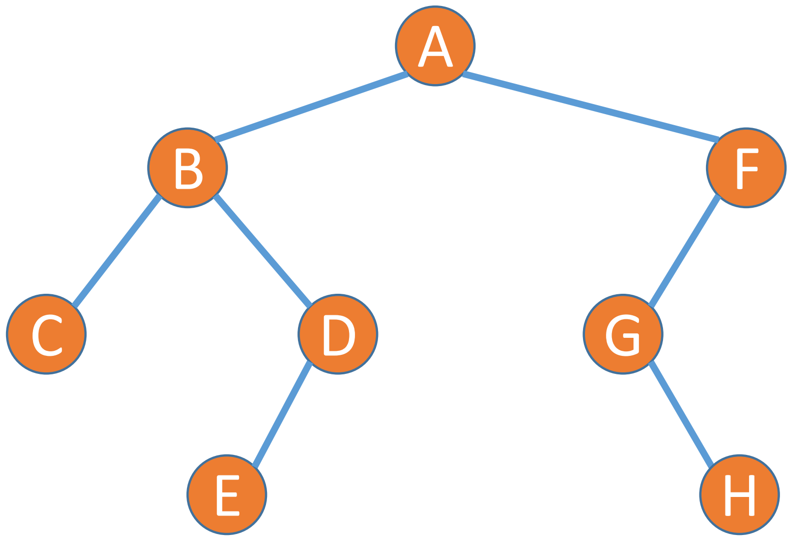
1. 将根结点进队；

2. 队不空时循环：从队列中出列一个结点*p，访问它；

   * 若它有左孩子结点，将左孩子结点进队；

   * 若它有右孩子结点，将右孩子结点进队。

6.4 二叉树的层次遍历



A　B　F　C　D G　E　H

A	**B**	**F**	**C**	**D**	**G**	**E**	**H**		

6.4 二叉树的层次遍历

使用队列类型定义如下：

```
Typedef struct{
    BTNode data[MaxSize];    //存放队中元素
    int front, rear;         //队头和队尾指针
}SqQueue;                    //顺序循环队列类型
```

6.4 二叉树的层次遍历

二叉树层次遍历算法：

```
void LevelOrder(BTNode *b){
    BTNode *p;       SqQueue *qu;
    InitQueue(qu);                                  //初始化队列
    enQueue(qu, b);                                 //根结点指针进入队列
    while(!QueueEmpty(qu) ) {                       //队不为空，则循环
        deQueue(qu, p);                             //出队结点p
        printf("%c", p -> data);                    //访问结点p
        if(p -> lchild != NULL)    enQueue(qu, p -> lchild);       //有左孩子时将其进队
        if(p -> rchild != NULL)    enQueue(qu, p -> rchild);       //有右孩子时将其进队
    }
}
```

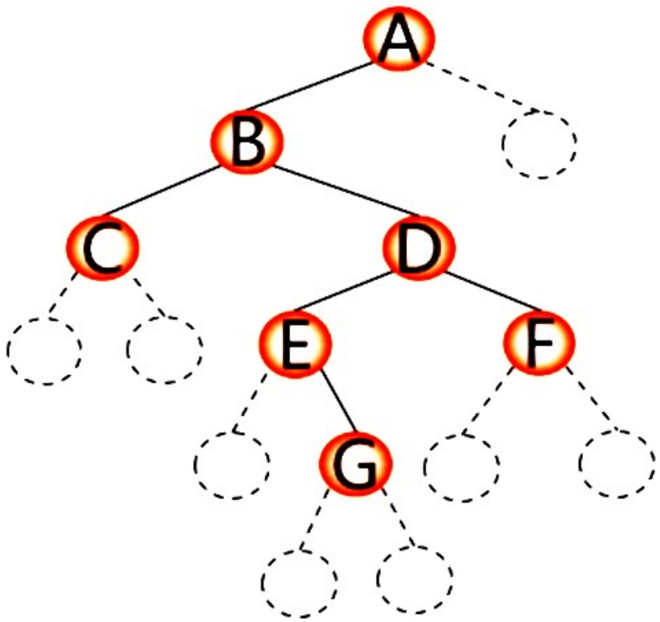
6.4 二叉树遍历算法的应用-二叉树的建立

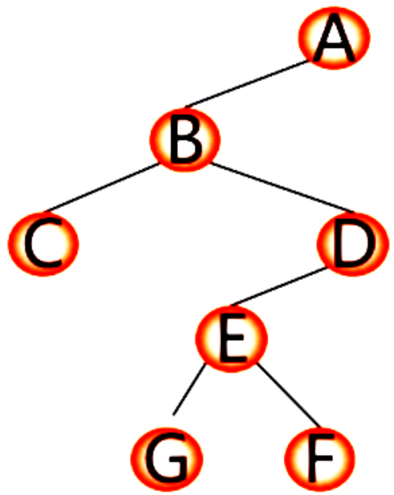
按先序遍历序列建立二叉树的二叉链表

例：已知先序序列为：ABCDEGF

（1）从键盘输入二叉树的结点信息，建立二叉树的存储结构

（2）在建立二叉树的过程中按照二叉树先序方式建立；





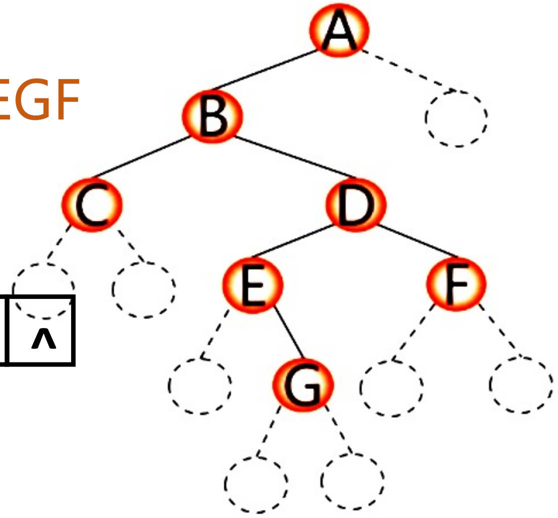
6.4 二叉树遍历算法的应用-二叉树的建立

杭州電子科技大学 HANGZHOU DIANZI UNIVERSITY

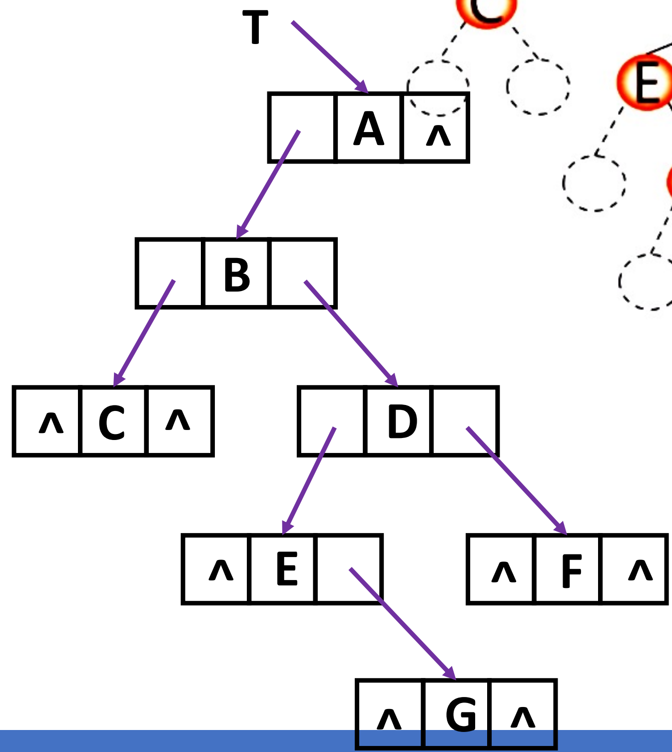
对右图所示二叉树，按下列顺序读入字符：

ABCΦΦDEΦGΦΦFΦΦΦ

ABCDEGF

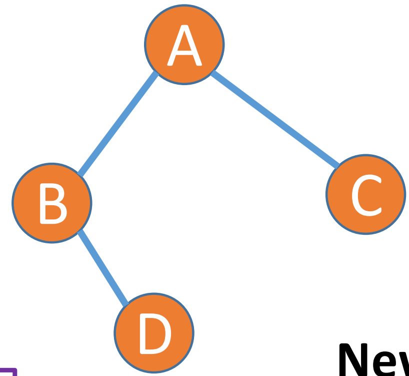



```
Status CreateBiTree(BiTree &T){
   scanf(&ch);
   if (ch == ' ') T = NULL;
   else {
      if (!(T = (BiTNode *) malloc (sizeof(BiTNode))))
          exit (OVERFLOW);
      T->data = ch; //生成根节点
      CreateBiTree (T->lchild); //构造左子树
      CreateBiTree (T->rchild); //构造右子树
    }
    return OK;
}
```

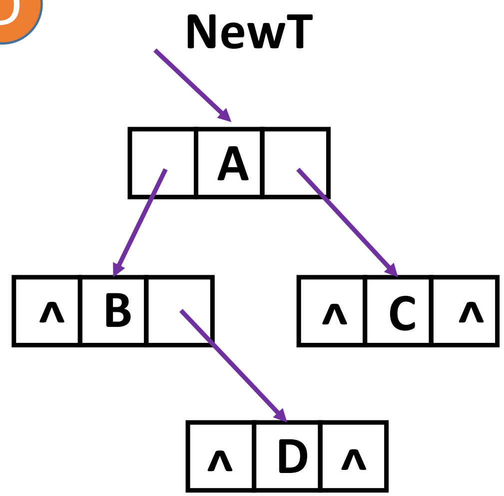


6.4 二叉树遍历算法的应用-复制二叉树

- 如果是空树，递归结束；
- 否则，申请新结点空间，复制根结点
  - 递归复制左子树
  - 递归复制右子树



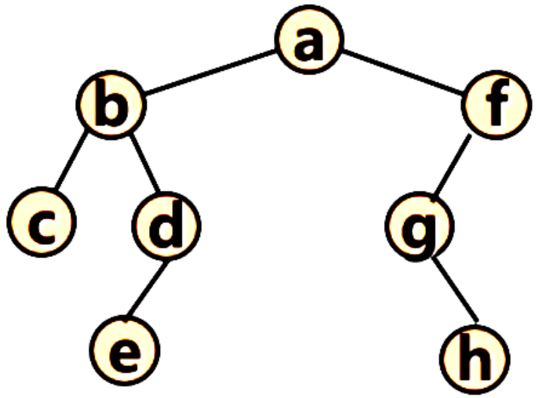

```
int Copy(BiTree T, BiTree &NewT){
   if(T==NULL) {    //如果是空树返回0
        NewT=NULL;  return 0;
    }
   else {
        NewT=new BiTNode;        NewT->data=T->data;
        Copy(T->lChild,  NewT->lchild);
        Copy(T->rChild,  NewT->rchild);
} }
```



6.4 二叉树遍历算法的应用-计算二叉树深度

- 如果是空树，则深度为0；
- 否则，递归计算左子树的深度记为m，递归计算右子树的深度记为n，二叉树的深度则为m与n的较大者加1.

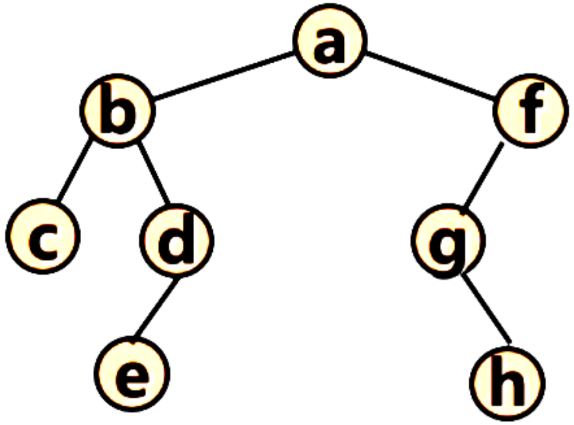
```
int Depth( BiTree T){
   if(T==NULL)      return 0;  //如果是空树返回0
   else {
         m=Depth(T->lChild);
         n =Depth(T->rChild);
         if(m>n) return (m+1);
         else return(n+1);
     }
}
```



6.4 二叉树遍历算法的应用-计算二叉树结点总数

- 如果是空树，则结点个数为0；
- 否则，结点个数为左子树的结点个数 + 右子树的结点个数再 + 1

```
int NodeCount(BiTree T){
   if(T == NULL )
     return 0;
   else
     return NodeCount(T->lchild)+
              NodeCount(T->rchild)+1;
}
```

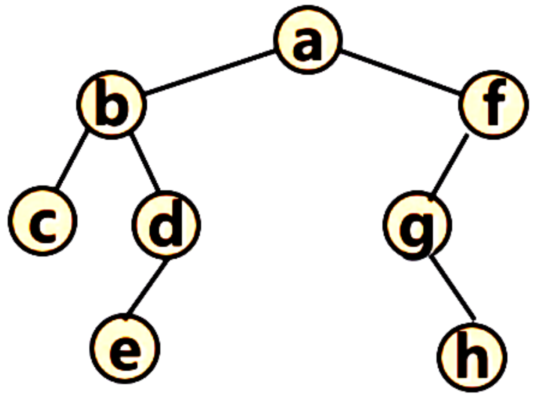


6.4 二叉树遍历算法的应用-计算二叉树叶子结点数（补充）

杭州電子科技大学 HANGZHOU DIANZI UNIVERSITY

- 如果是空树，则叶子结点个数为0；
- 否则，为左子树的叶子结点个数 + 右子树的叶子结点个数。

```
int LeadCount(BiTree T){
    if(T==NULL)       //如果是空树返回0
        return 0;
    if (T->lchild == NULL && T->rchild == NULL)
        return 1;     //如果是叶子结点返回1
    else
        return LeafCount(T->lchild) +
               LeafCount(T->rchild);
}
```



第6章 树和二叉树

6.1 树和二叉树的定义

- 案例引入

6.2 二叉树的抽象数据类型定义

6.3 二叉树的性质和存储结构

6.4 遍历二叉树和线索二叉树

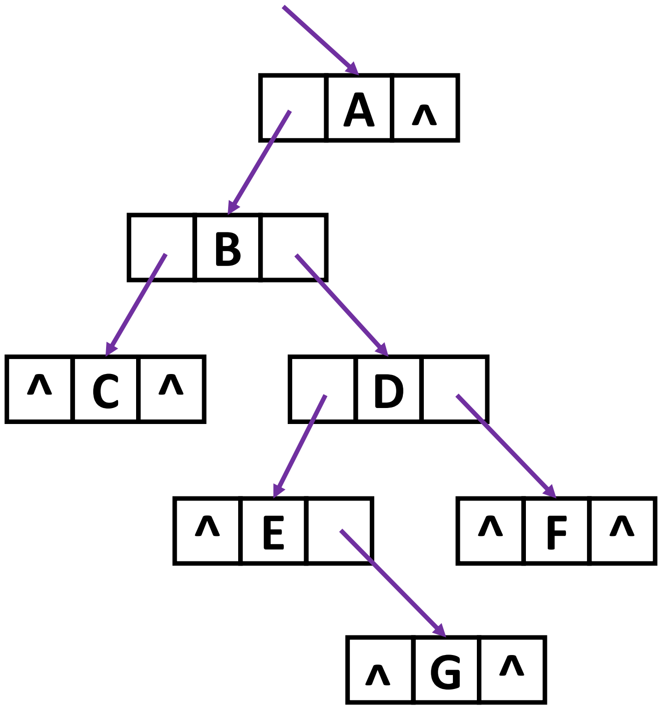
6.5 树和森林

6.6 哈夫曼树及其应用

6.6 案例分析与实现

6.4 线索二叉树

问题：为什么要研究线索二叉树?



当用二叉链表作为二叉树的存储结构时，可以很方便地找到某个结点的左右孩子；但一般情况下，无法直接找到该结点在某种遍历序列中的前驱和后继结点。

6.4 线索二叉树

提出的问题：

如何寻找特定遍历序列中二叉树结点的前驱和后继？

解决的方法：

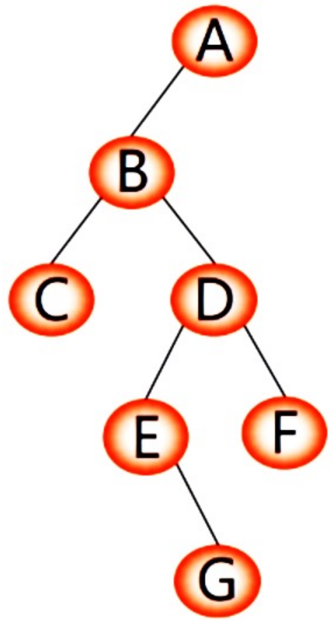
1. 通过遍历寻找——费时间
2. 再增设前驱、后继指针域——增加了存储负担
3. 利用二叉链表中的空指针域

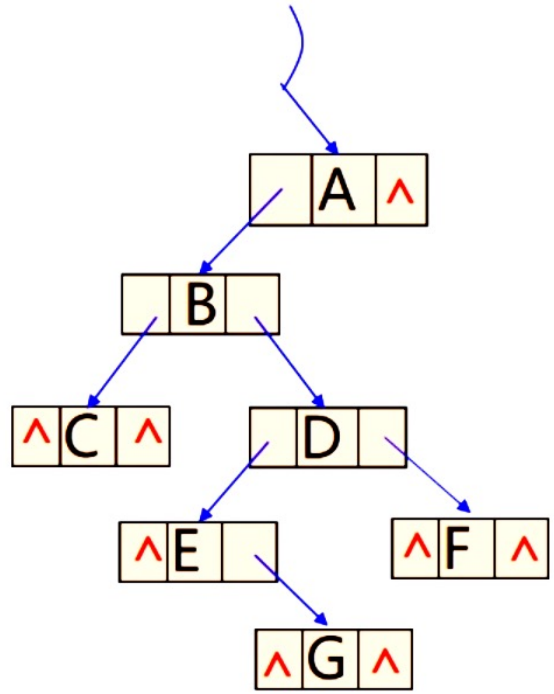
6.4 线索二叉树

回顾：

二叉树链表中空指针域的数量：

具有n个结点的二叉链表中，一共有2n个指针域；因为n个结点中有n-1个孩子，即2n个指针域中，有n-1个用来指示结点的左右孩子，其余n+1个指针域为空。





6.4 线索二叉树

杭州电子科技大学 HANGZHOU DIANZI UNIVERSITY

利用二叉链表中的空指针域：

如果某个结点的左孩子为空，则将空的左孩子指针域改为指向其前驱；如果某结点的右孩子为空，则将空的右孩子指针域改为指向其后继。

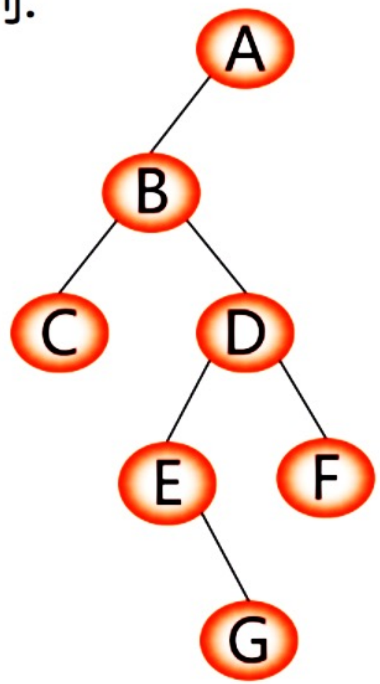
——这种改变指向的指针称为“线索”

加上了线索的二叉树称为线索二叉树（Threaded Binary Tree）

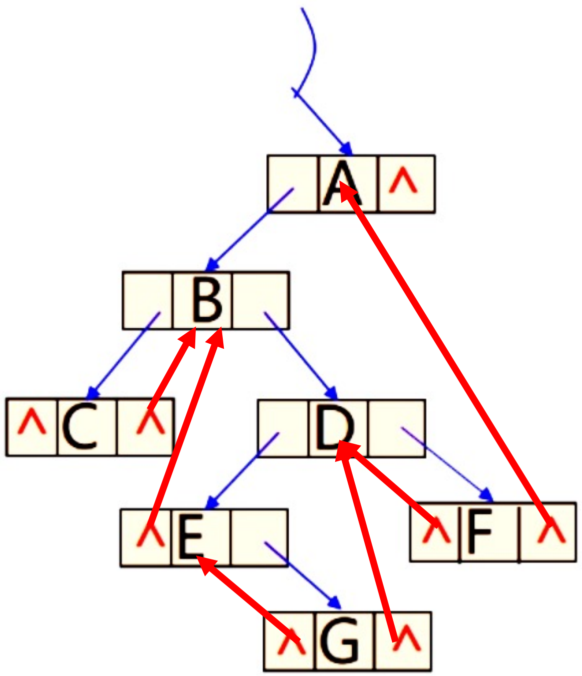
对二叉树按某种遍历次序使其变为线索二叉树的过程叫线索化

6.4 线索二叉树

例:



中序遍历：CBEGDFA



6.4 线索二叉树

杭州電子科技大学 HANGZHOU DIANZI UNIVERSITY

为区分lchild和rchild指针到底是指向孩子的指针，还是指向前驱或者后继的指针，对二叉链表中每个结点增设两个标志域ltag和rtag，并约定：

ltag = 0	lchild 指向该结点的左孩子
ltag = 1	lchild 指向该结点的前驱
rtag = 0	rchild 指向该结点的右孩子
rtag = 1	rchild 指向该结点的后继

6.4 线索二叉树

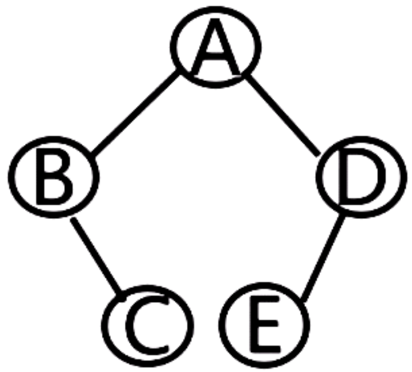
这样，结点的结构为：

lchild	ltag	data	rtag	rchild

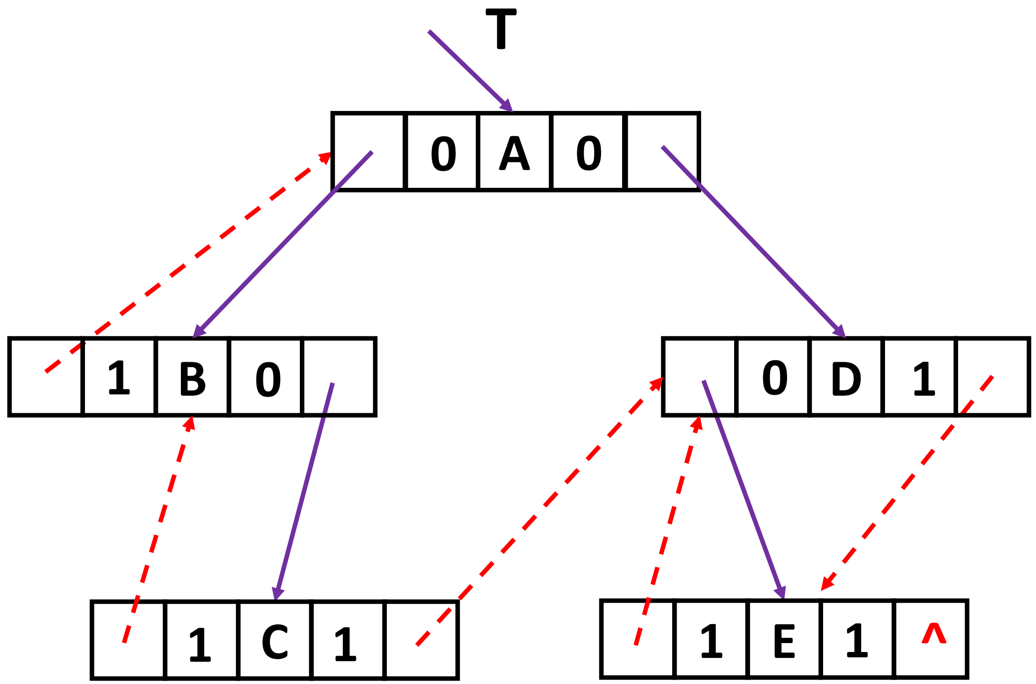
```
typedef struct BiThrNode{
    TElemType data;
    int ltag, rtag;         //左右标志
    struct BiThrNode  *lchild, *rchild;    //左右孩子指针
}BiThrNode, *BiThrTree;
```

6.4 线索二叉树

先序线索二叉树

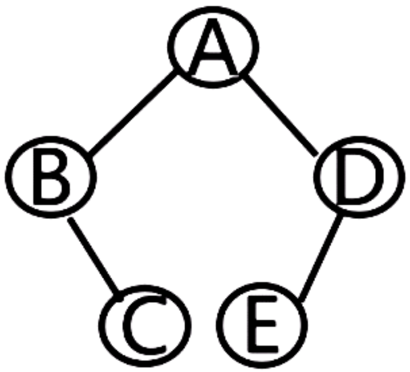


先序序列：A B C D E

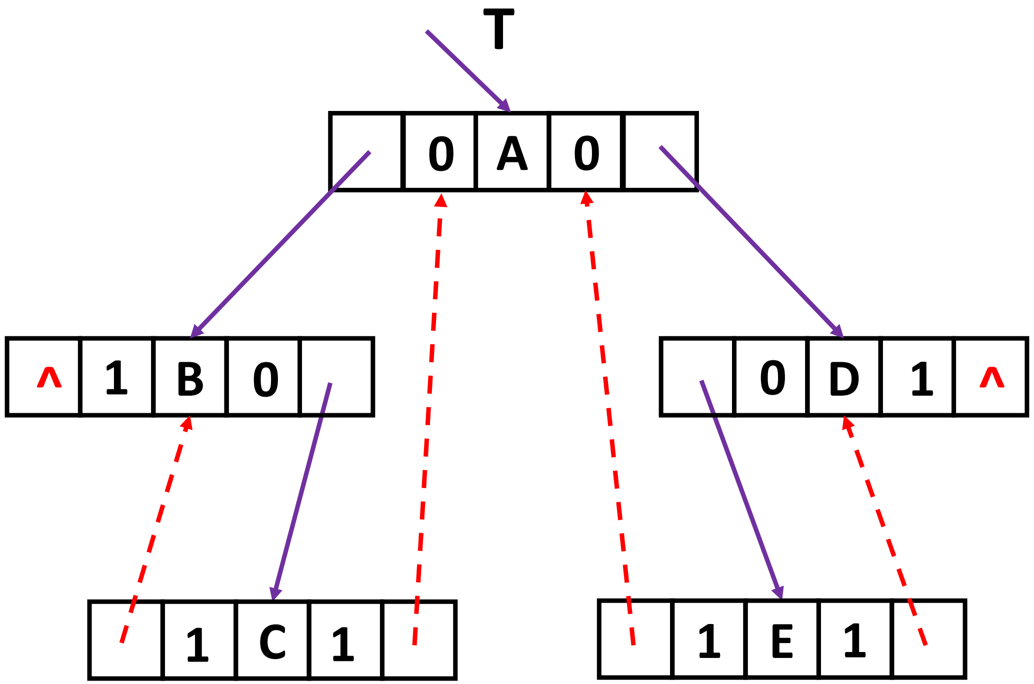


6.4 线索二叉树

中序线索二叉树

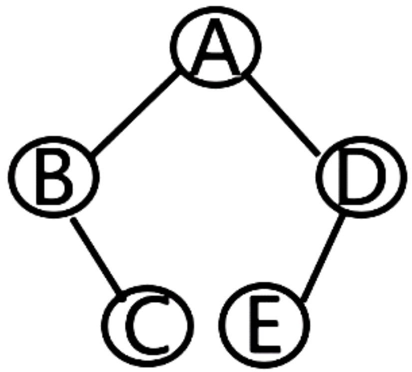


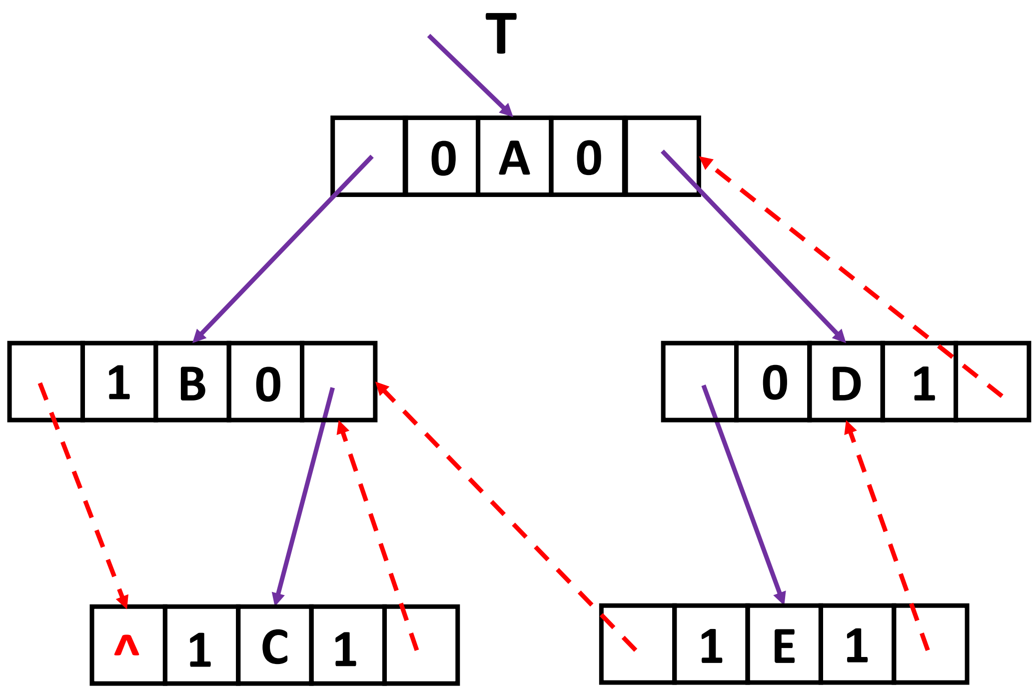
中序序列：B C A E D



6.4 线索二叉树

后序线索二叉树



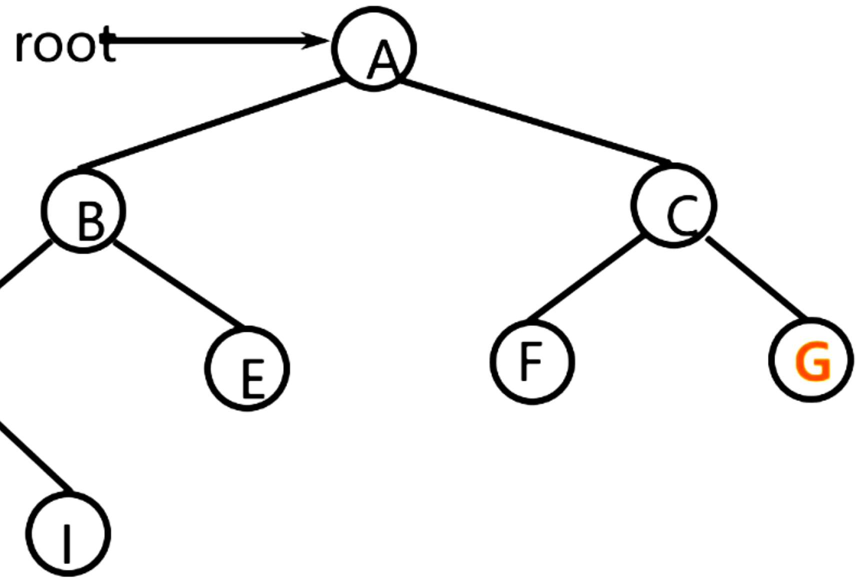


后序序列：C B E D A

6.4 线索二叉树

练习：画出以下二叉树对应的中序线索二叉树。

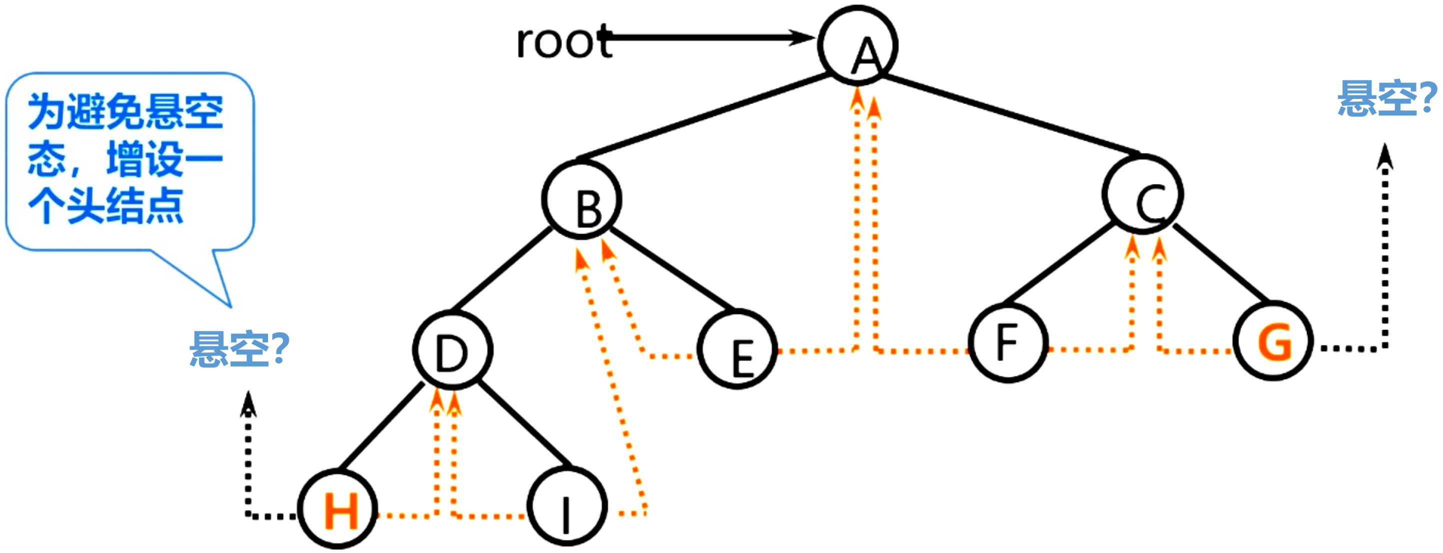
该二叉树中序遍历结果为：H, D, I, B, E, A, F, C, G



6.4 线索二叉树

练习：画出以下二叉树对应的中序线索二叉树。

该二叉树中序遍历结果为：H, D, I, B, E, A, F, C, G



6.4 线索二叉树

杭州电子科技大学 HANGZHOU DIANZI UNIVERSITY

增设了一个头结点

ltag=0, lchild指向根结点,

rtag=1, rchild指向遍历序列中最后一个结点

遍历序列中第一个结点的lc域和最后一个结点的rc域都指向头结点